Sissejuhatus infotehnoloogiasse

Operatsioonisüsteemid

Sisukord

Opsüsteemide eesmärgid ja põhifunktsioonid

Opsüsteemide ajalugu

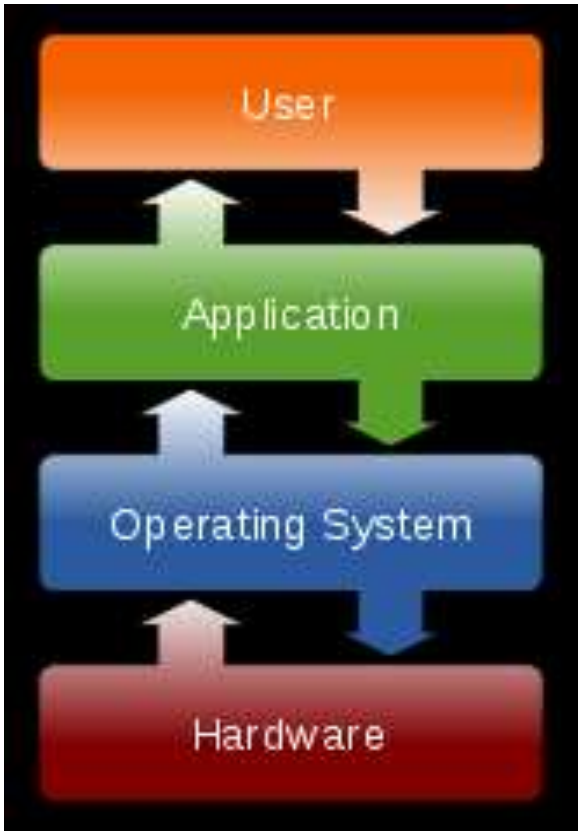
Suured praegu levinud opsüsteemid

Opsüsteemide struktuur

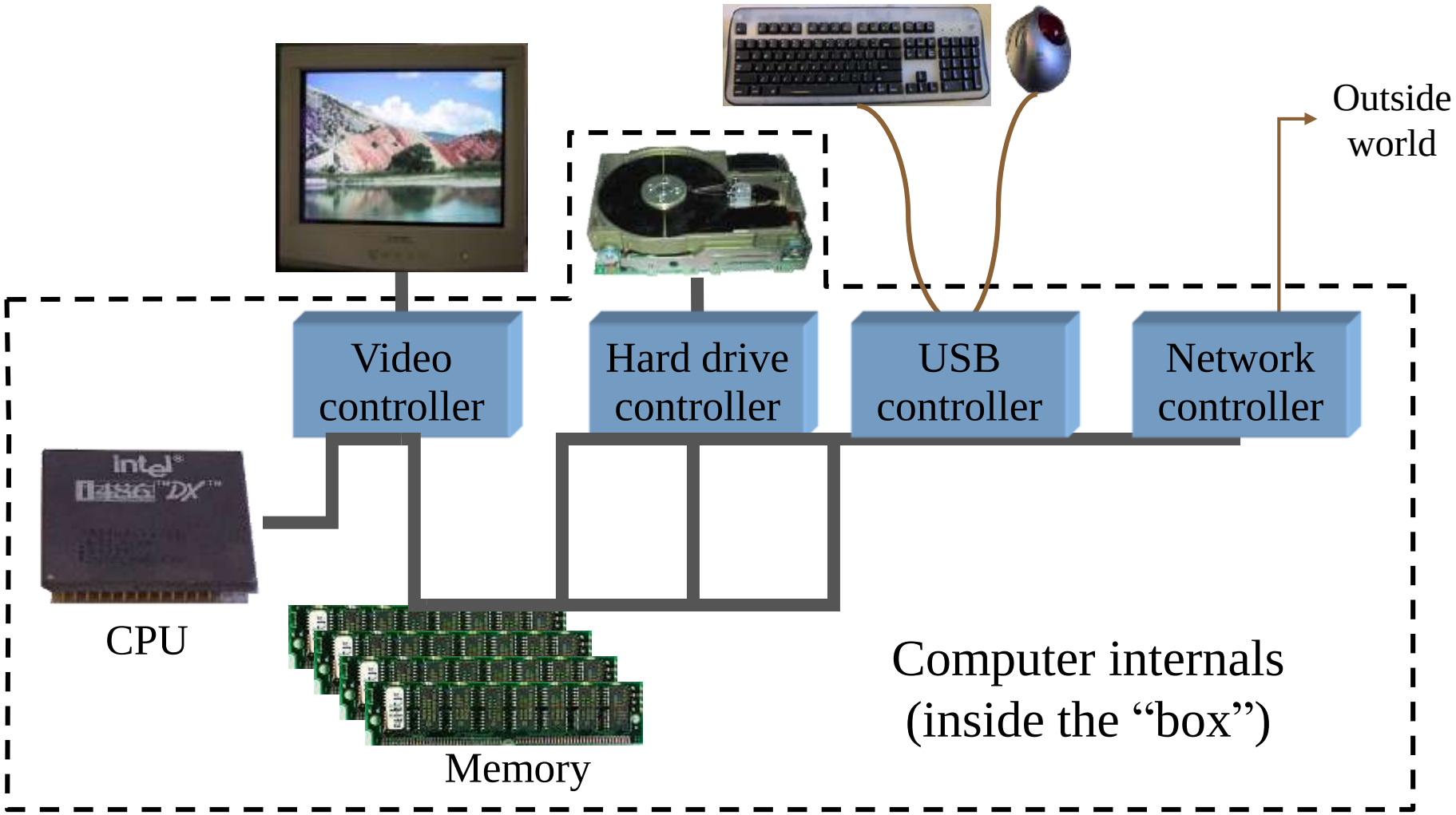
Opsüsteemi tuuma tegevus

Failisüsteemid

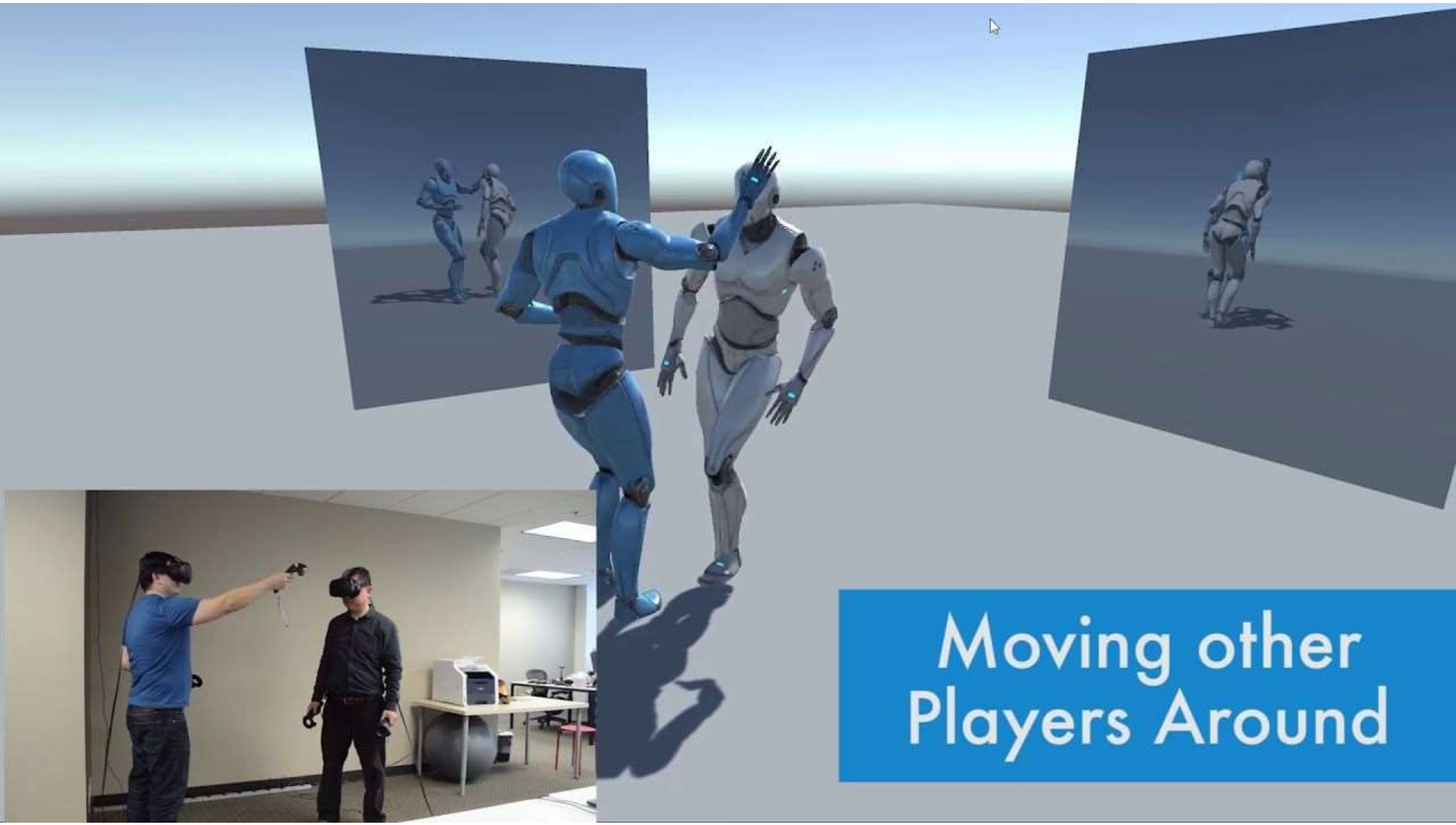
Virtualiseerimine



Components of a simple PC



OS ehitab programmide jaoks simuleeritud virtuaalmaailma



Miks opsüsteem?

Opsüsteemi põhieesmärgid:

- Pakkuda programmeerijale valmistehtud standardtükke.
- Võimaldada kasutajal arvutis ühtemoodi ja harjumuspäraselt tegutseda, sõltumatult sellest, mis programmid tal arvutis on.

Arvutit saaks programmeerida ka ilma opsüsteemita. Sel juhul:

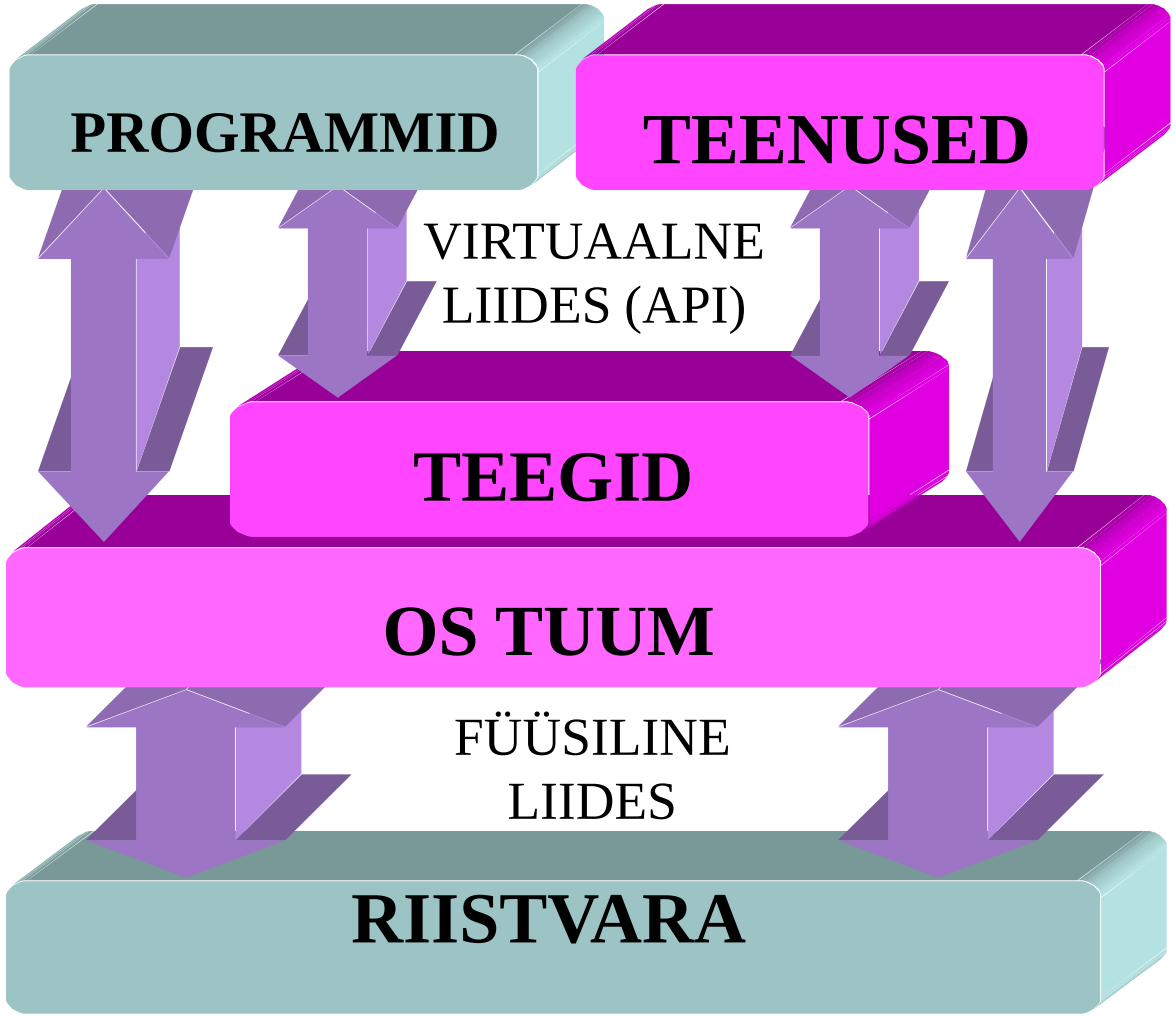
- Oleks iga programmi tegemine palju raskem
- Kasutajate jaoks näeks eri programmid väga eri moodi välja.

Mida opsüsteem teeb ja teha oskab?

- Oskab kettalt programme lugeda ja neid käima panna.
- Oskab programme seisma panna
- Oskab anda programmidele parasjagu aega ja mälu
- Oskab programme vajadusel korraks peatada ja taaskäivitada
- Oskab kettale faile ja kataloge kirjutada ja sealt neid lugeda.
- Oskab välisseadmetega (printer, monitor jne) suhelda.
- Oskab võrguga suhelda.
- Oskab intelligentselt mälu ja cachega tegeleda
- Oskab suhelda graafikakaardiga
- jne

Kui opsüsteemi ei oleks, peaks iga programm kõiki neid asju ise teha oskama!

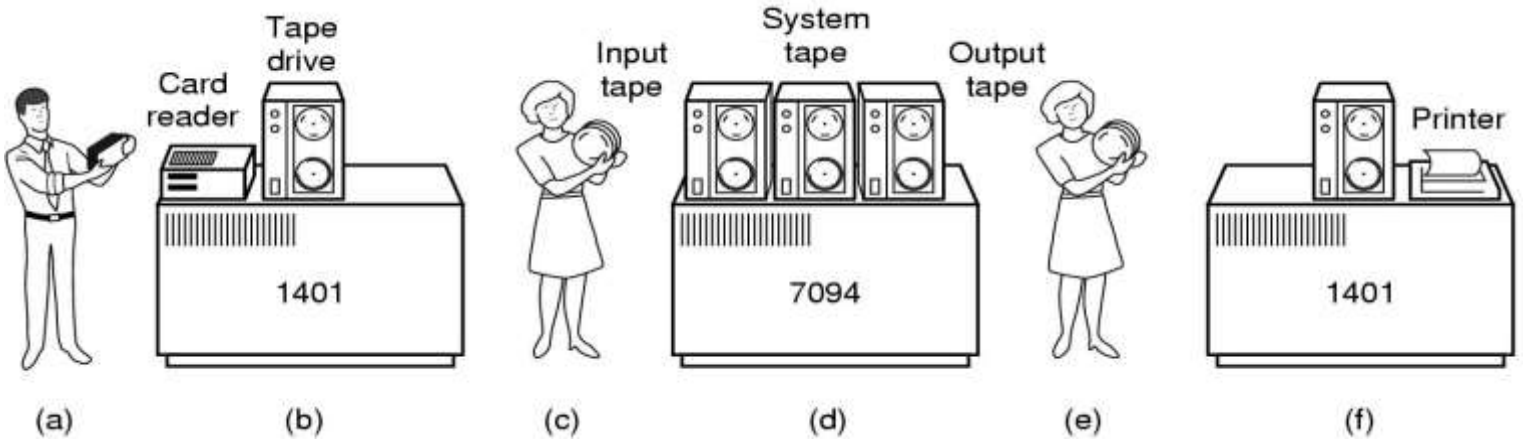
Operatsioonisüsteemi roll



Operating system history timeline

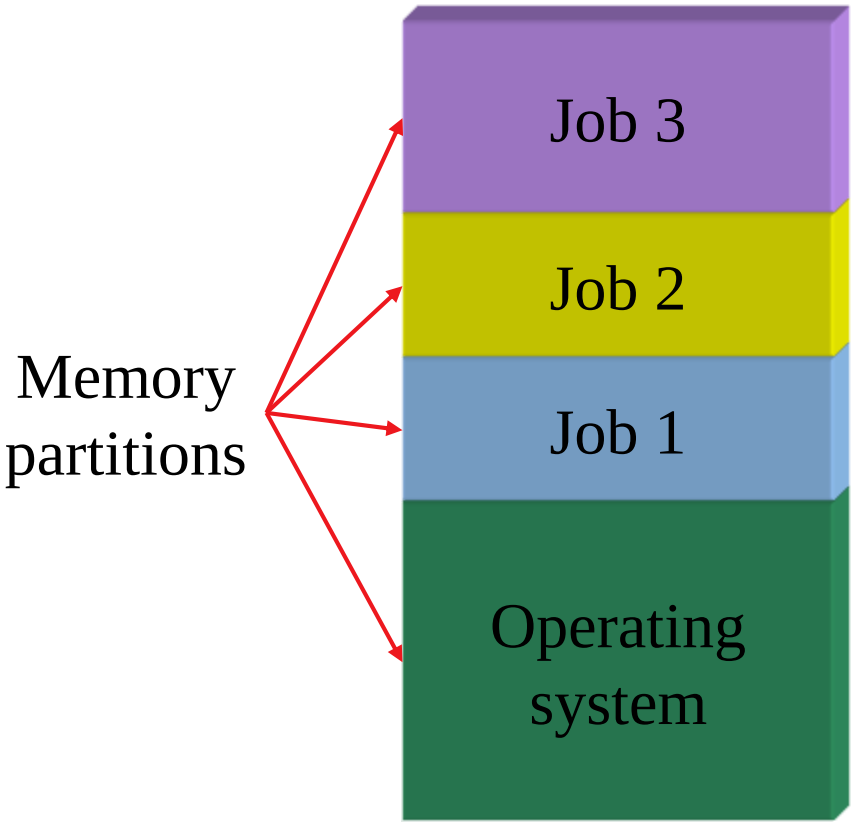
- **First generation**: 1945 – 1955
  - Vacuum tubes
  - Plug boards
- **Second generation**: 1955 – 1965
  - Transistors
  - Batch systems
- **Third generation**: 1965 – 1980
  - Integrated circuits
  - Multiprogramming
- **Fourth generation**: 1980 – present
  - Large scale integration
  - Personal computers
- Fifth generation: ?

Second generation: batch systems



- Bring cards to 1401
- Read cards onto input tape
- Put input tape on 7094
- Perform the computation, writing results to output tape
- Put output tape on 1401, which prints output

Third generation: multiprogramming



- Multiple jobs in memory
  - Protected from one another
- Operating system protected from each job as well
- Resources (time, hardware) split between jobs
- Still not interactive
  - User submits job
  - Computer runs it
  - User gets results minutes (hours, days) later

Natuke ajalugu

1969	AT&T Bell Labs UNIX
1977	BSD UNIX
1979	AT&T UNIX kommertskasutusse
1981	MS-DOS
1983	SCO UNIX
1984	SunOS
1987	OS/2
1987	MINIX (avatud koodiga õppe-opsüsteem)
1989	AT&T UNIX SVR4 (System V Release 4), POSIX
1991	Linux
1992	SUSE linux
1993	Windows NT
1993	Debian Linux
1994	RedHat Linux
2000	Windows 2000
2001	OS X (Mac UNIXil), praeguse nimega macOS
2007	iOS (baseerub OS X-l + lihtsustusi + täiendusi)
2008	Android (Linux + Java + palju lisasid ja täiendusi)

Suured praegused opsüsteemid, nende põhirakendusvaldkonnad ja umbkaudsed turuosad 2024

- **Windows** (Microsoft)
  - Tava-arvutid (73%)
  - Serverid (25% servereid)
- **macOS ja iOS** (Apple)
  - Mobiilid (iphone 28% / ipad 55 %)
  - Tava-arvutid (mac, 15%)
- **Linux** (vabavara)
  - Mobiilid (Android, 72% mobiile)
  - Serverid (63% servereid)
  - Pisiseadmed (domineeriv)
  - Tava-arvutid (4.5 % linux, 2.25 % chromeOS)
- **BSD** (vabavara)
  - Serverid

Mõned väiksemad:

- Symbian (mobiil)
- Blackberry (mobiil)
- Bada (Samsungi mobiil)
- TinyOS (pisiseadmed)
- zOS (IBM mainframes)
- arcaOS (OS/2 derivative)
- Haiku (beOS derivative)
- FreeDOS (open-source DOS)

Unix

Some IBM PC OS-s related to windows

Historical:

PC-DOS-2000 is a text-based desktop operating system made by IBM as an update of the older MS-DOS.

OS/2 is a „high performance“ desktop and “high end“ operating system made by IBM (started jointly with Microsoft, latter branched into NT)

Current:

ReactOS is a free OS, binary compatible with Windows XP

Wine is a free compatibility layer for Linuxes for running windows programs

DOSBox is a free MS-DOS & old IBM-PC emulator for windows and Linux

OS-es: Apple: historical

Macintosh OS 9, OS 8, OS 7, and OS 6 are desktop operating systems made by Apple Computers that first ran on Motorola 680x0 and later on Motorola/IBM PowerPC

Darwin is an open-source UNIX-based operating system that includes capabilities from BSD unix, the NeXT and Macintosh operating systems, with Mach (CMU) as the kernel base. Darwin was a foundation for OS X of the Macintosh.

masOS (previously OS X) is a desktop operating system based on Darwin. Macintosh OS X is made by Apple Computers and ran originally on Motorola/IBM PowerPC, then on Intel, now on ARM

iOS is a mobile operating system based on OS X. Runs on ARM family of processors.

OS-es: main free UNIXes

All these UNIXes are actively developed:

- **LINUX** is a free version of UNIX that runs on Intel/AMD, ARM and a large number of exotic processors
- **Android** is a free version/extension of Linux developed by Google and Open Handset Alliance, runs on ARM and Intel
- **FreeBSD**, **NetBSD** and **OpenBSD** are different BSD-based free UNIX versions
- **GNU Hurd** is a free UNIX-based operating system, based on Mach (CMU) BSD-type microkernel

OS-es: commercial UNIXes

Alive and kicking commercial UNIXes:

- **macOS and iOS** are Apple operating systems based on Mach and various BSD unixes plus NextStep.

Practically dead commercial UNIXes

- **Solaris** is a UNIX-based operating system made by Sun Computers that runs on Sun SPARC and Intel Pentium
- **Sun-OS** is an older text-based UNIX that runs on Sun SPARC. Solaris is an enhancement of Sun-OS that includes a graphic user interface.
- **AIX** is IBM's version of UNIX.
- **HP-UX** is a UNIX-based operating system made by Hewlett-Packard that runs on HP PA RISC.
- **ULTRIX** is a UNIX-based operating system made by DEC. ULTRIX was later replaced by Digital UNIX.

Linux distributions aka distros: what is a distro?

Linux itself is just a kernel.

Compiled kernel size varies, from ca 1 to 50 megabytes.

The largest part of the kernel source is a huge collection of drivers.

The Linux system requires a large set of libraries and tools on top of the kernel. This is mostly Gnu software. Hence **Gnu/Linux**.

There are ca 300 different distros. Distros offer:

- Pre-compiled kernel, libraries and tools, ready to install from image
- A set of ready-made and packaged software (editors, browsers, desktop software) in the initial installation package
- An easy and configurable set of tools to install more software
- A large set of pre-configured software packages the installation tool will pull from the net and install

Linux distro tree

Distros are typically built on each other: a new distro uses an old one, modifies some features, adds some, removes some.

A small number of distros are „core distros“ on top of what many others are built.

See: https://en.wikipedia.org/wiki/List_of_Linux_distributions

Important core distros:

Debian: .deb packages, focus on fully free software
RedHat: largest commercial Linux provider, .rpm packages
Slackware: one of the earliest, focus on stability and simplicity
Gentoo: software installed by building from source
Arch: minimalist, geared towards expert hackers

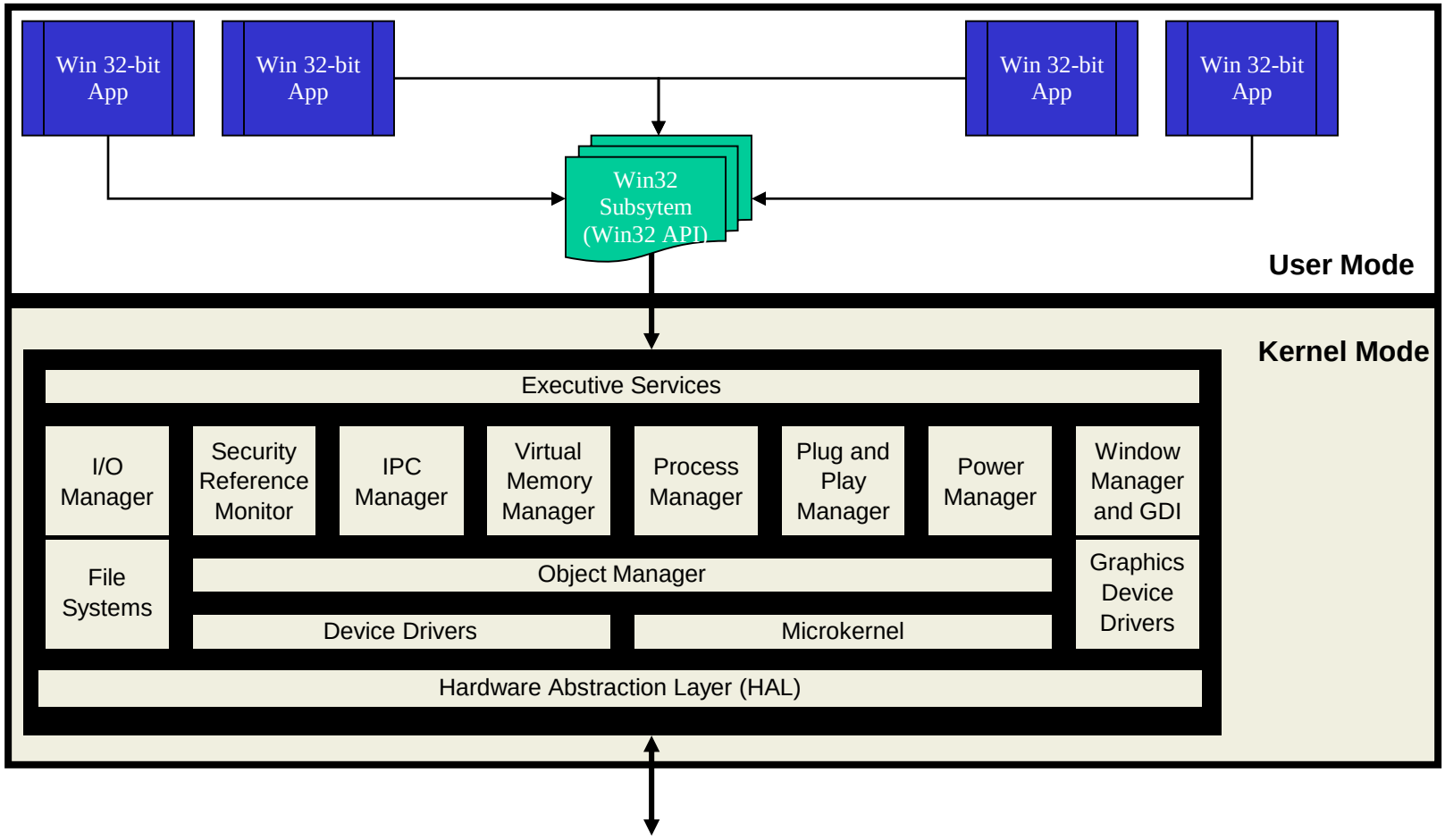
Popular linux distros

First distro still alive: **slackware**

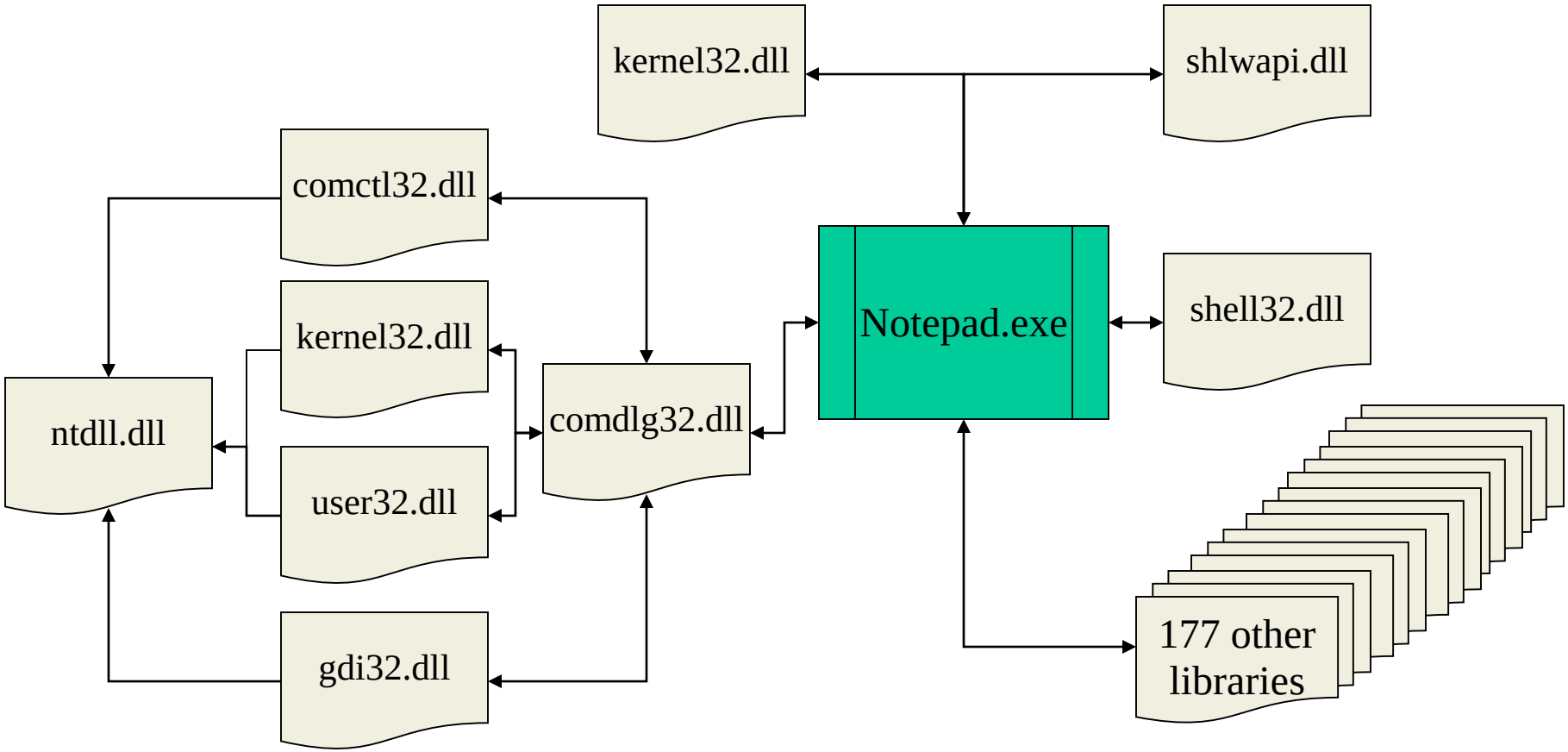
Some of the most popular ones:

Mint: based on Ubuntu, offers special versions of Gnome 2 and 3 desktop
Mageia: a freeware version of commercial Mandriva (RedHat derivative)
Ubuntu: based on Debian, offers Unity fork of Gnome desktop
Fedora: a freeware version of commercial RedHat
OpenSuse: freeware version of SuseLinux (Slackware derivative)
Debian: conservative distro wildly popular on servers, base for many others
Arch: original, minimalist
PCLinuxOS (Mandriva derivative)
CentOS (RedHat derivative)
Zorin (Ubuntu derivative)
... 61. **Gentoo** (source-based) ...

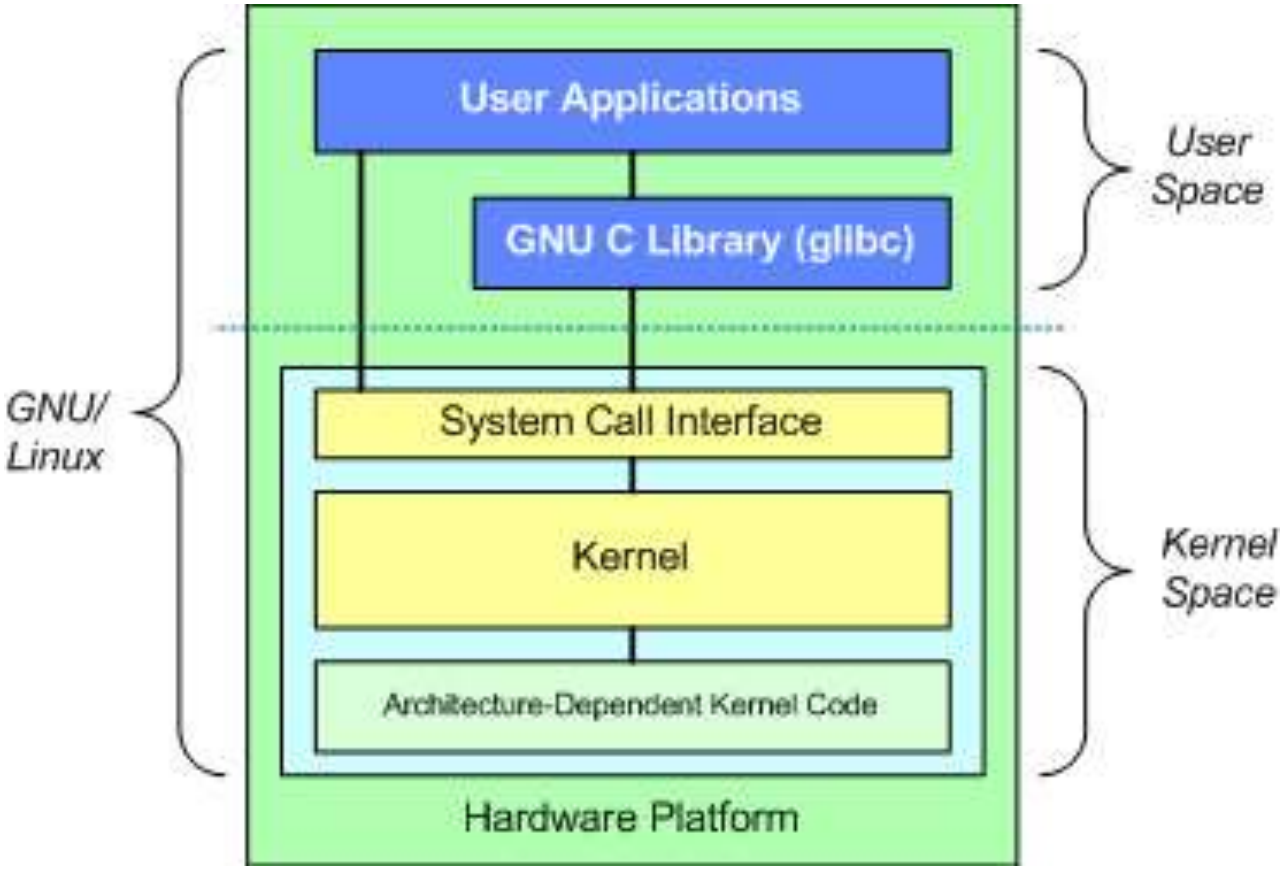
Windows structure high-level view



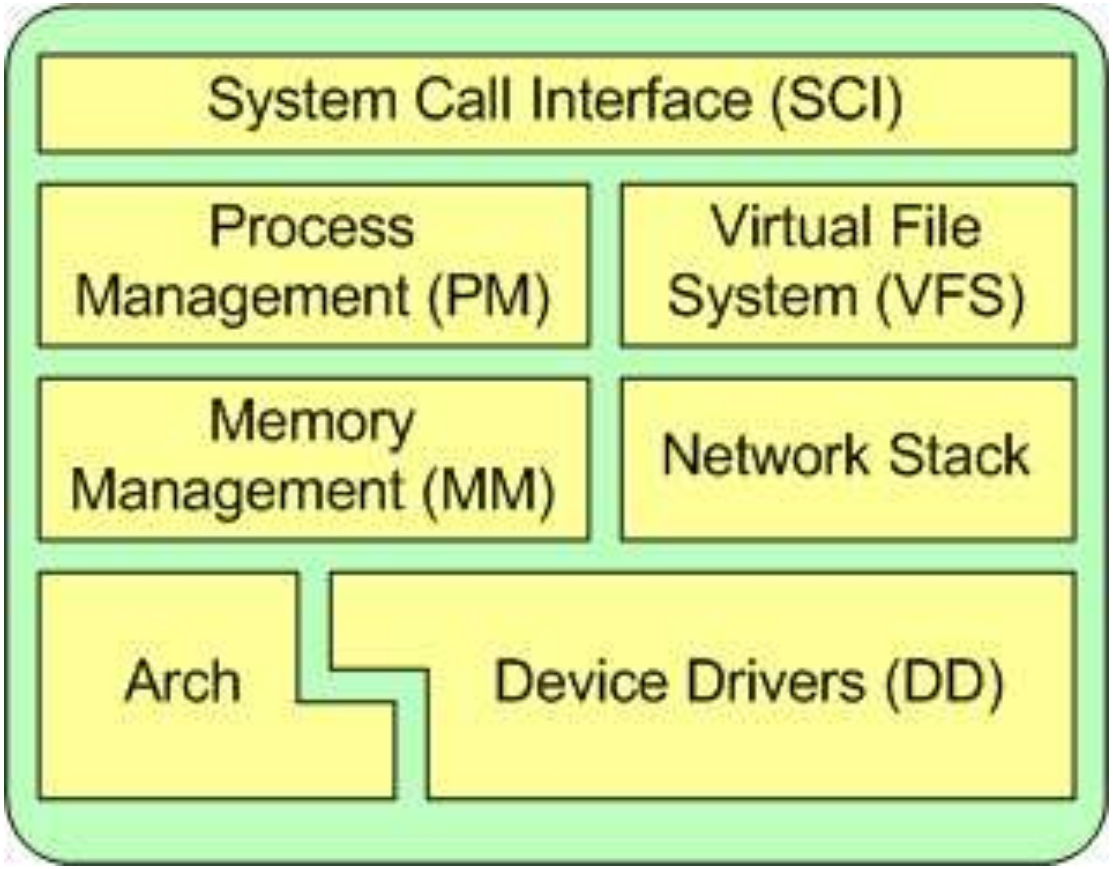
Opening a file in notepad: 180 dynamic link libraries



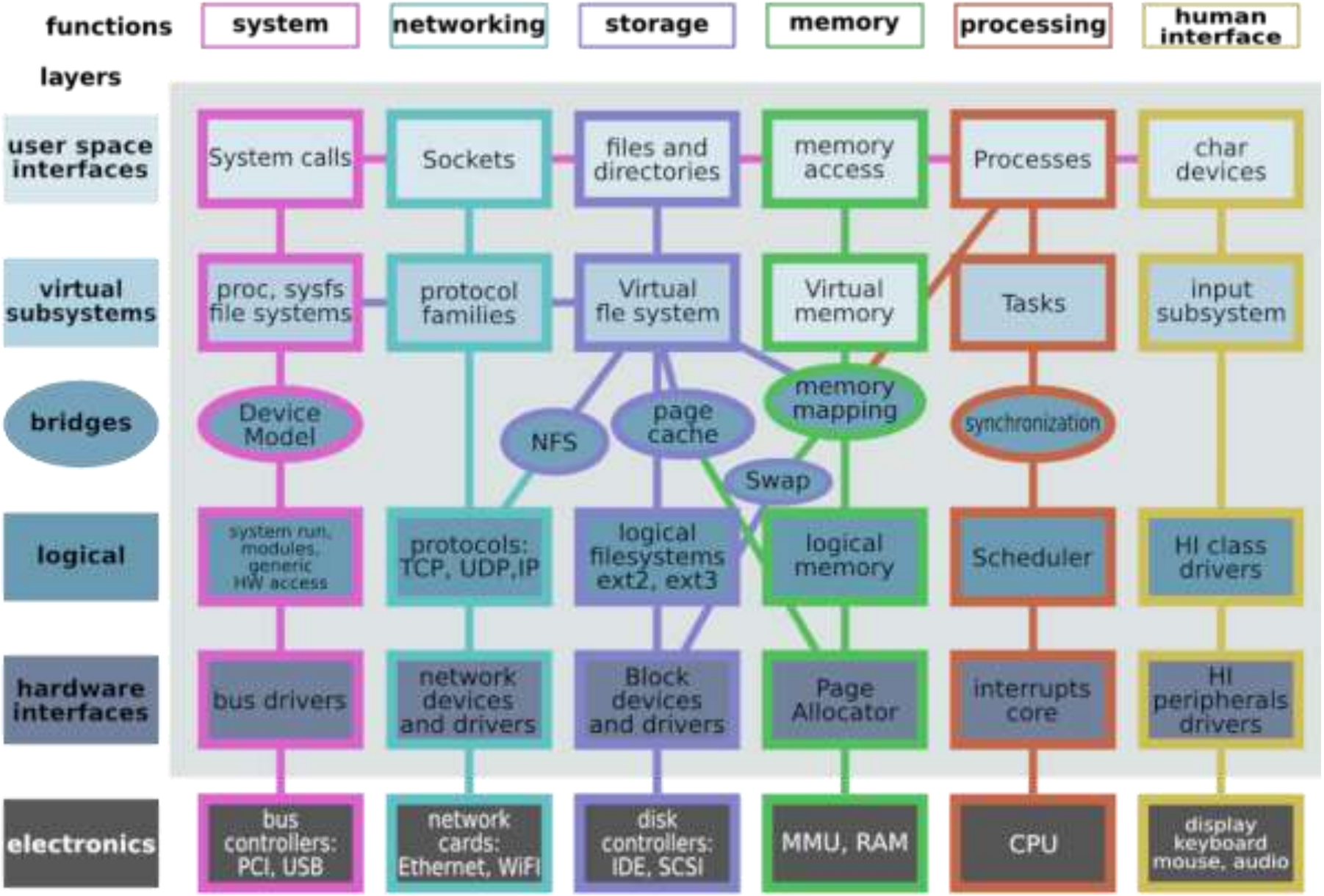
Linux structure: high-level view



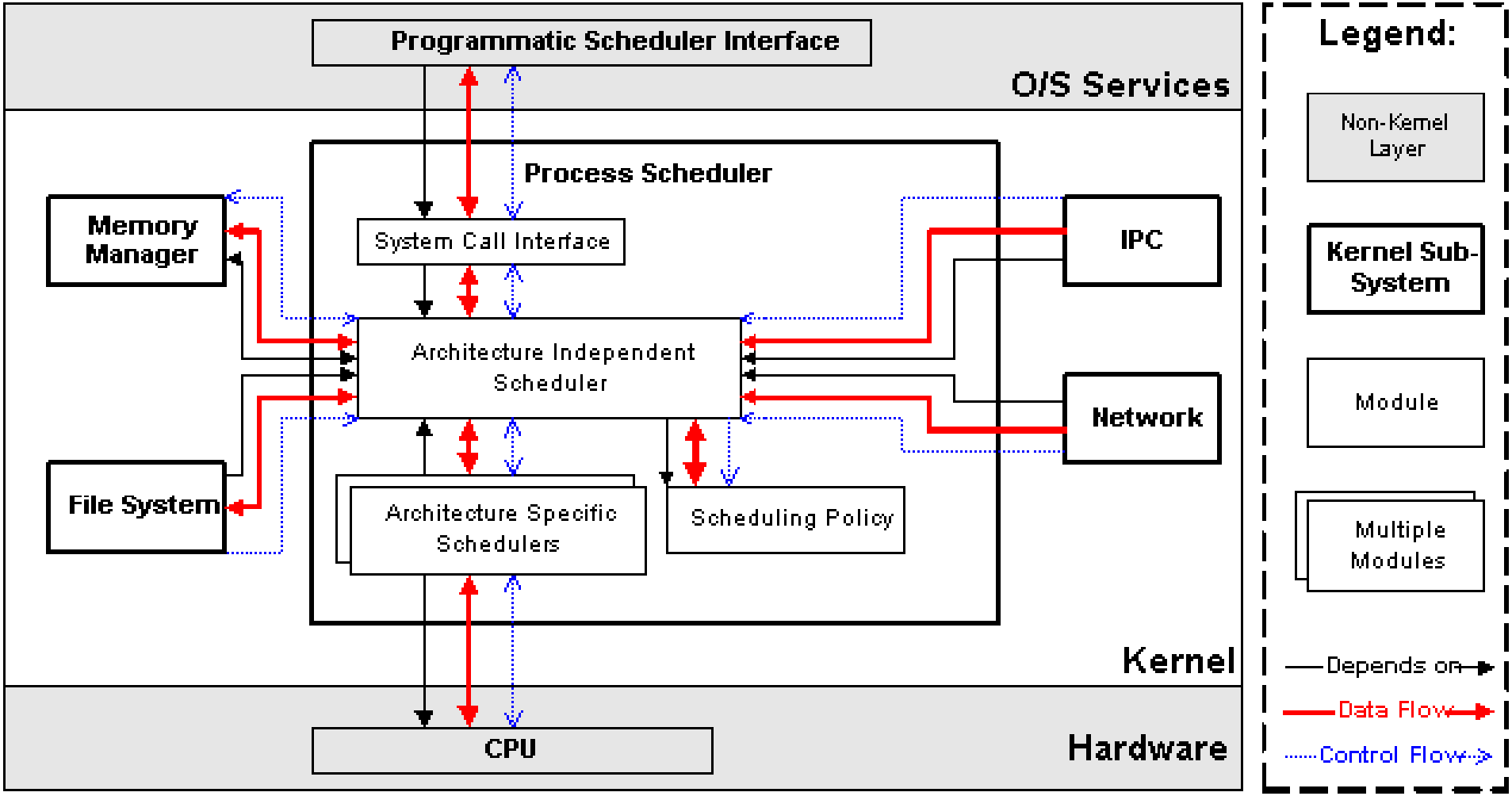
Linux kernel structure high-level view:



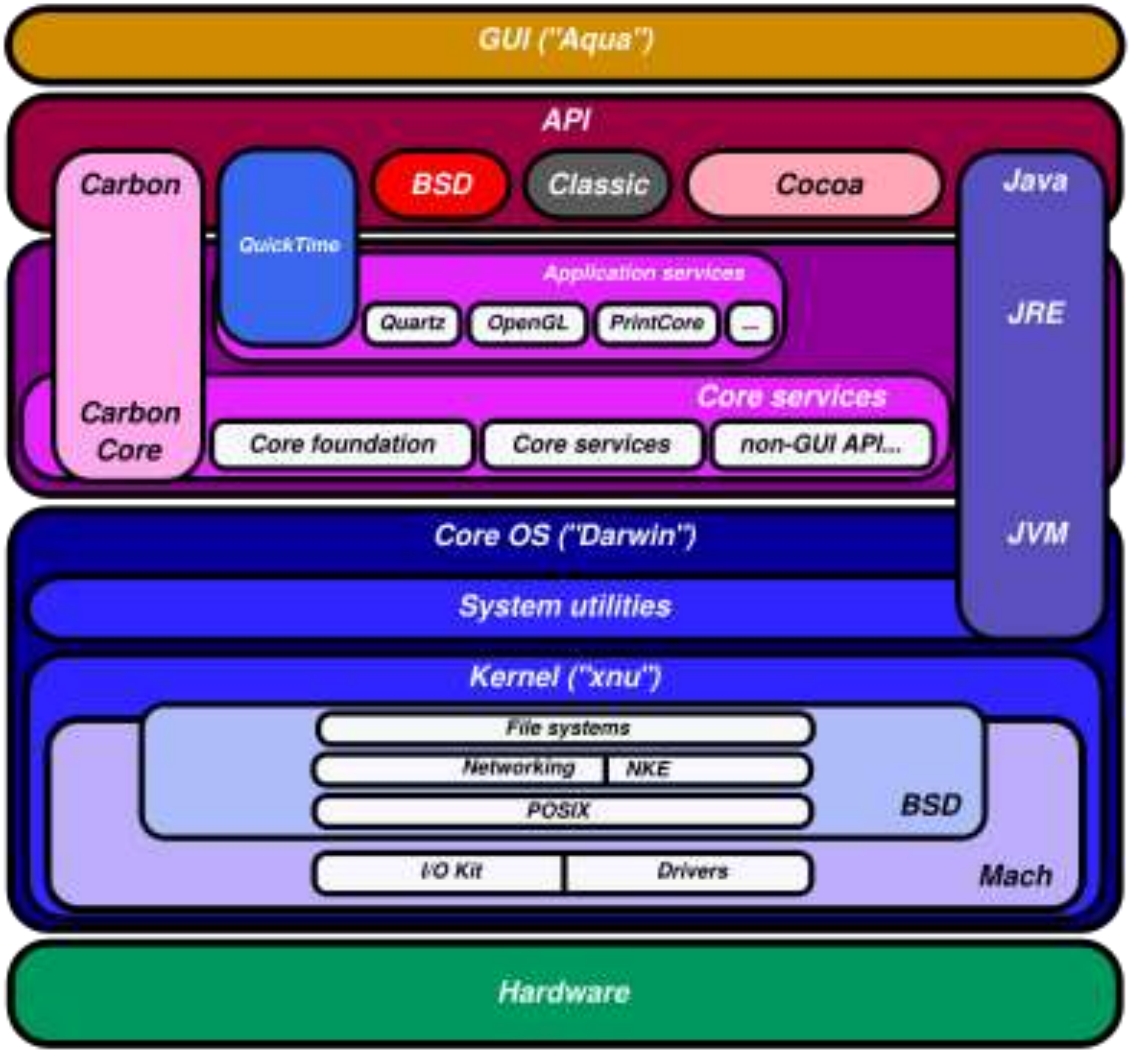
Linux kernel diagram



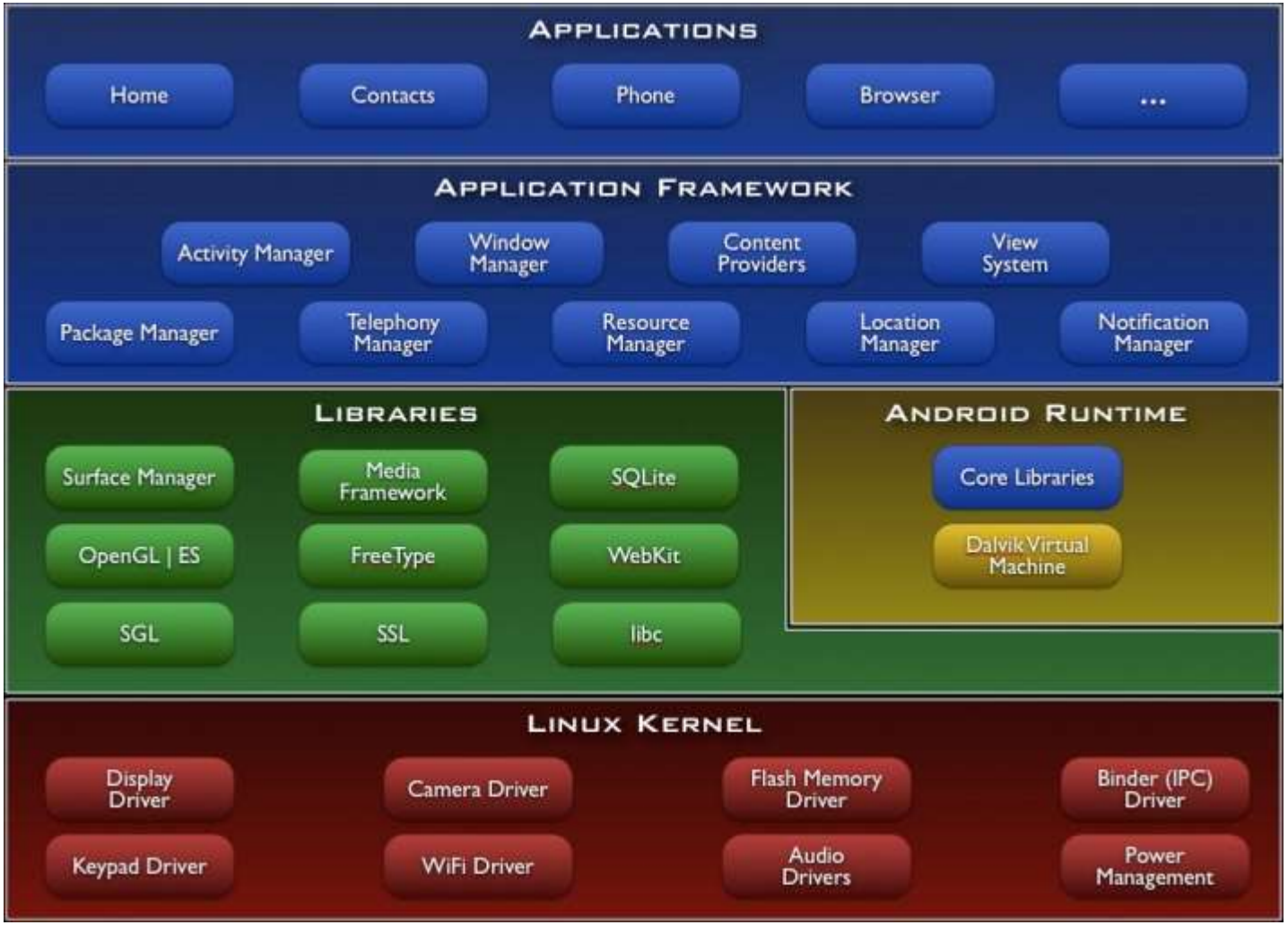
Linux structure: scheduler-centric view

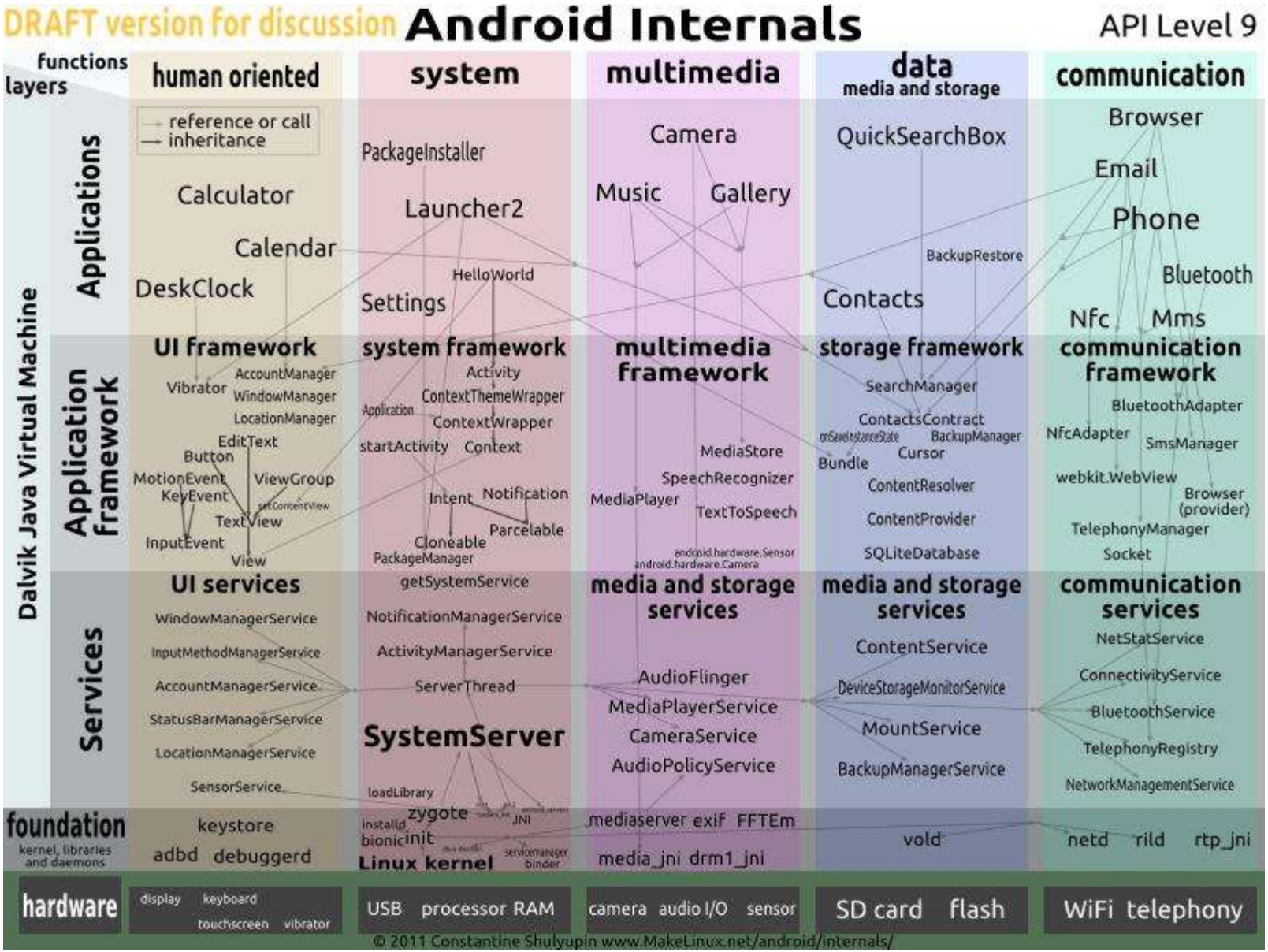


MacOS structure

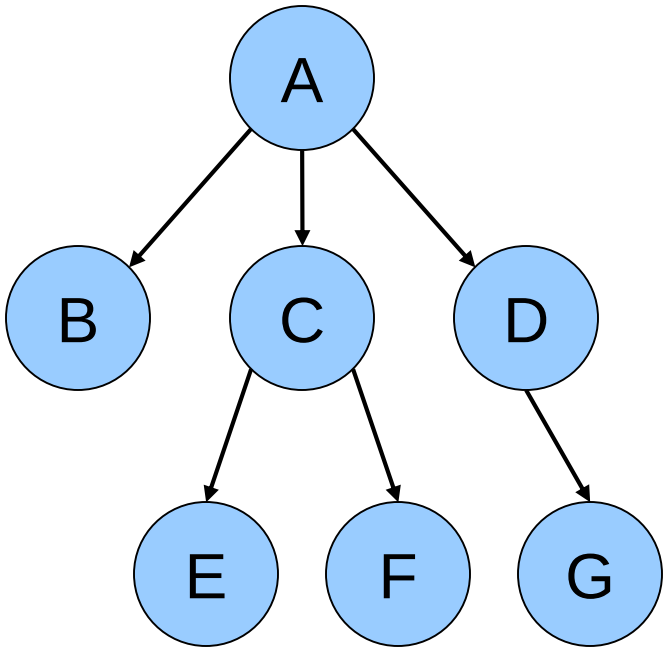


Android OS structure




DRAFT version for discussion
Android Internals
API Level 9
functions
layers
human oriented
system
multimedia
data
media and storage
communication
reference or call
inheritance
Applications
Calculator
Calendar
DeskClock
PackageInstaller
Launcher2
HelloWorld
Settings
Camera
Music
Gallery
QuickSearchBox
BackupRestore
Contacts
Browser
Email
Phone
Bluetooth
Nfc
Mms
Dalvik Java Virtual Machine
Application framework
UI framework
AccountManager
Vibrator
WindowManager
LocationManager
EditText
Button
MotionEvent
ViewGroup
KeyEvent
setContentView
TextView
InputEvent
View
system framework
Activity
ContextThemeWrapper
Application
ContextWrapper
startActivity
Context
Intent
Notification
Parcelable
Cloneable
PackageManager
multimedia framework
MediaStore
SpeechRecognizer
MediaPlayer
TextToSpeech
android.hardware.Sensor
android.hardware.Camera
storage framework
SearchManager
ContactsContract
onSaveInstanceState
BackupManager
Cursor
Bundle
ContentResolver
ContentProvider
SQLiteDatabase
communication framework
BluetoothAdapter
NfcAdapter
SmsManager
webkit.WebView
Browser
(provider)
TelephonyManager
Socket
Services
UI services
WindowManagerService
InputMethodManagerService
AccountManagerService
StatusBarManagerService
LocationManagerService
SensorService
getSystemService
NotificationManagerService
ActivityManagerService
ServerThread
SystemServer
media and storage services
AudioFlinger
MediaPlayerService
CameraService
AudioPolicyService
media and storage services
ContentService
DeviceStorageMonitorService
MountService
BackupManagerService
communication services
NetStatService
ConnectivityService
BluetoothService
TelephonyRegistry
NetworkManagementService
foundation
kernel, libraries and daemons
keystore
adbd
debuggerd
loadLibrary
zygote
JNI
install.d
bionic
init
Linux kernel
servicemanager
binder
mediaserver
exif
FFTEm
media_jni
drm1_jni
vold
netd
rild
rtp_jni
hardware
display
keyboard
touchscreen
vibrator
USB
processor
RAM
camera
audio I/O
sensor
SD card
flash
WiFi
telephony
© 2011 Constantine Shulyupin www.MakeLinux.net/android/internals/

Processes

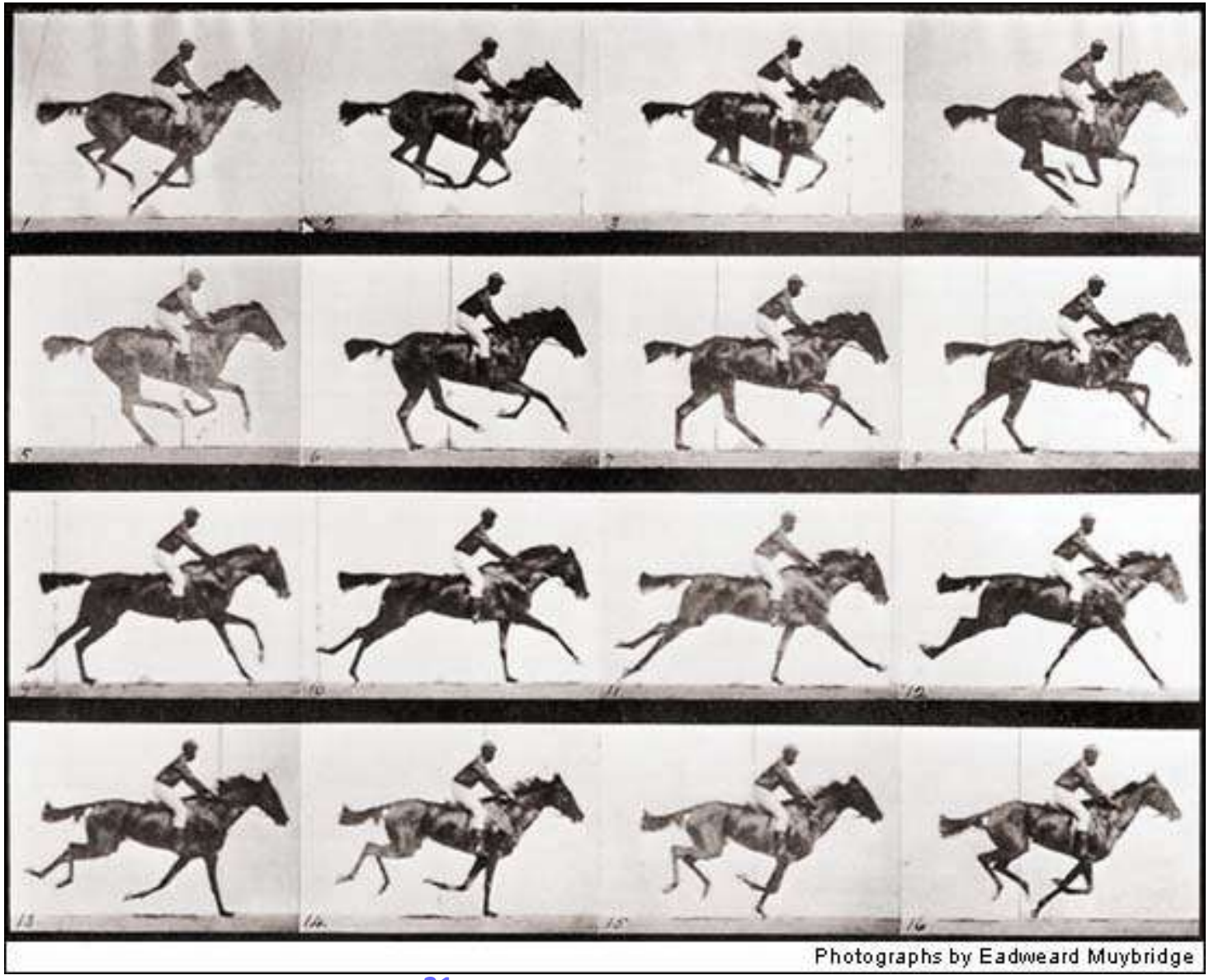


- Process: program in execution
  - Address space (memory) the program can use
  - State (registers, including program counter & stack pointer)
- OS keeps track of all processes in a *process table*
- Processes can create other processes
  - Process tree tracks these relationships
  - A is the *root* of the tree
  - A created three child processes: B, C, and D
  - C created two child processes: E and F
  - D created one child process: G

Multitasking and scheduling

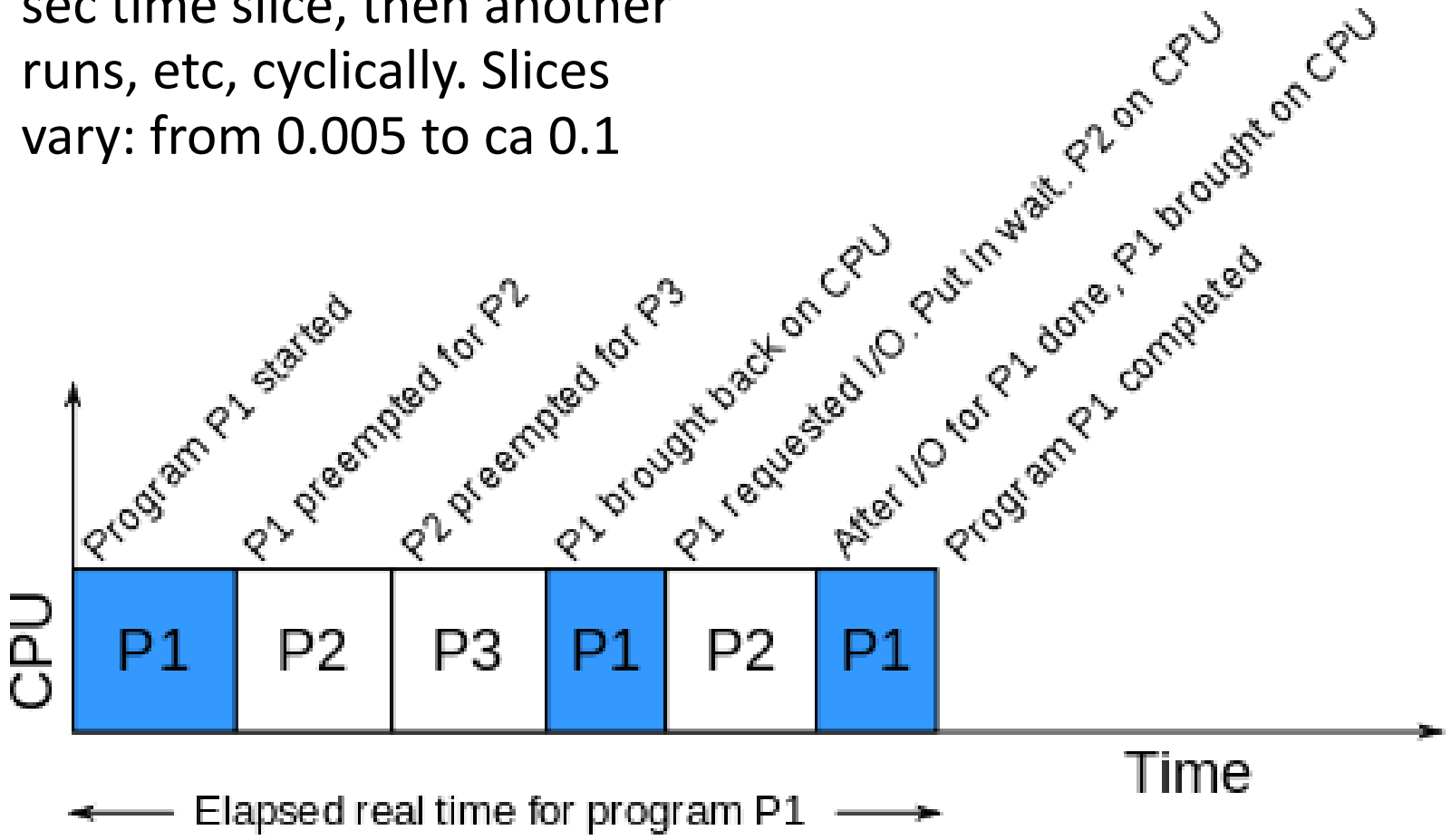
Parallel running of processes is a movie-like illusion

http://www.youtube.com/watch?v=qqsmei__kmQ

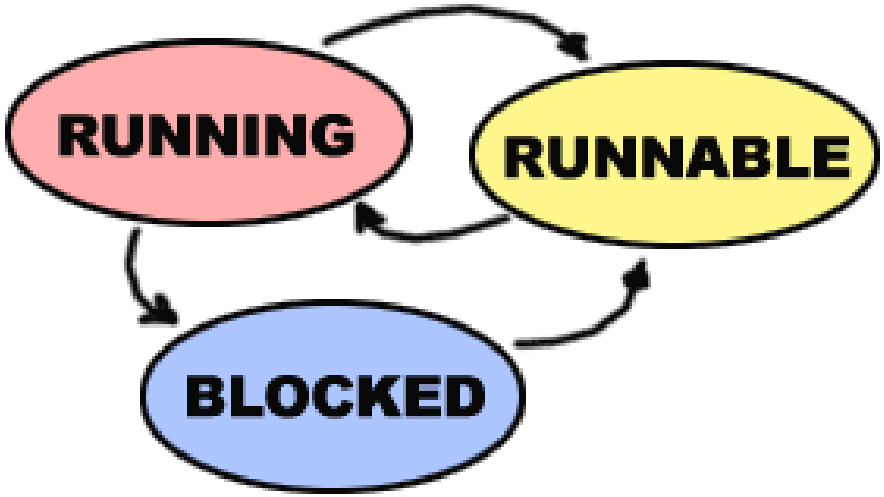


Running programs in parallel

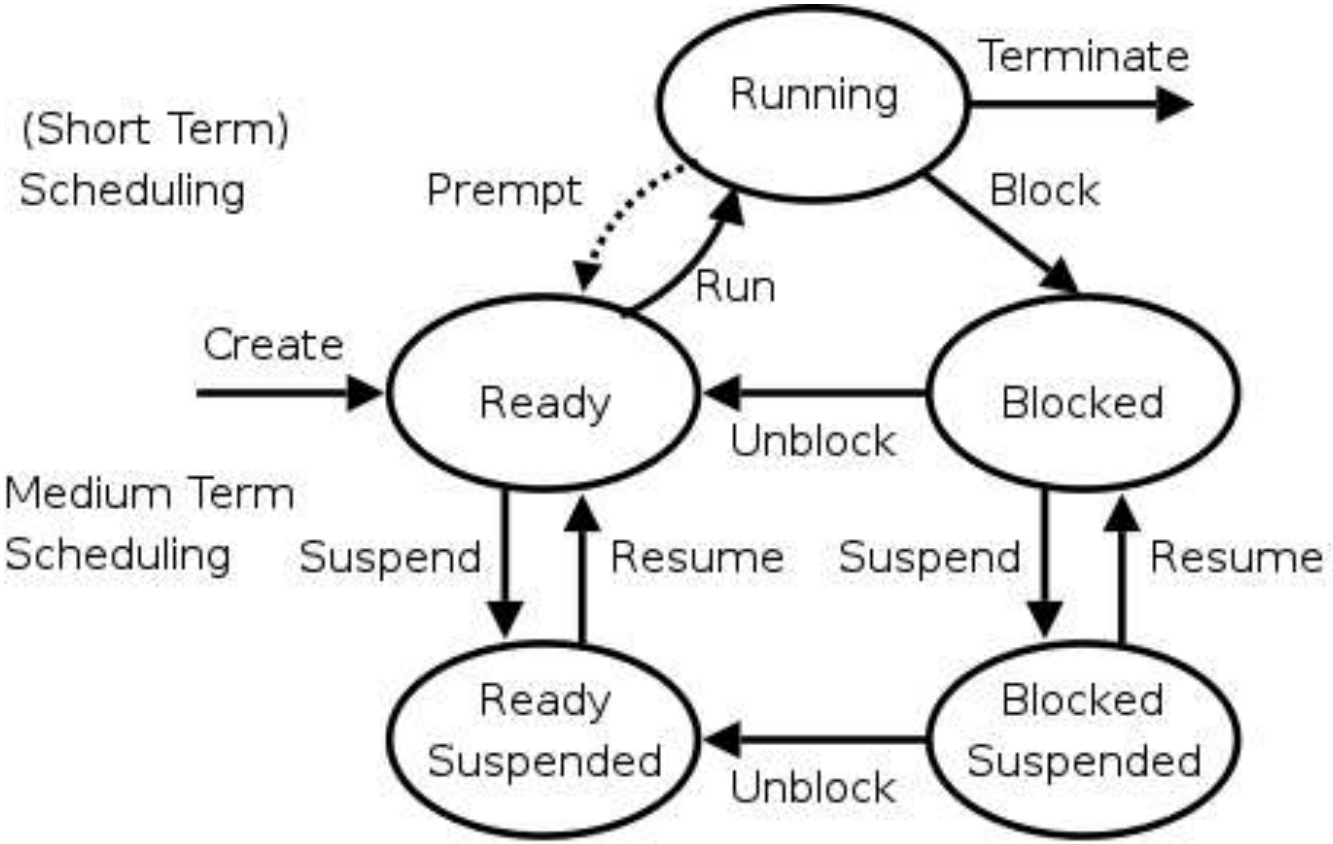
Each program gets ca 0.01 sec time slice, then another runs, etc, cyclically. Slices vary: from 0.005 to ca 0.1



Processes wait and run



Process lifecycle in more details



Unblock is done by another task (a.k.a. wakeup, release, V)
Block is a.k.a. sleep, request, P)

Main scheduling algorithm goals

Fairness.
Treating users fairly: everybody gets some time to run

Respecting priority.
That is, giving more important processes higher priorit

Efficiency.
Do not spend excessive time in the scheduler.
Try to keep all parts of the system busy.

Low turnaround time.
Minimize the time from the submission of a job to its termination.

High throughput.
Maximize the number of jobs completed per day.

Low response time.
Minimize the time from when an interactive user issues a command to when the response is given.

Repeatability.
Non-random, predictable behavious

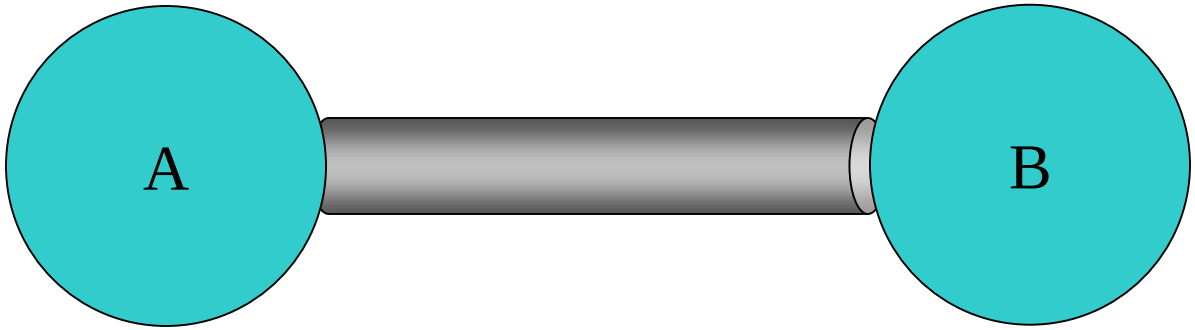
Degrade gracefully under load.

Scheduling algorithms: main versions

Scheduling algorithm	CPU Overhead	Throughput	Turnaround time	Response time
First In First Out	Low	Low	High	Low
Shortest Job First	Medium	High	Medium	Medium
Priority based scheduling	Medium	Low	High	High
Round-robin scheduling	High	Medium	Medium	High
Multilevel Queue scheduling	High	High	Medium	Medium

Interprocess communication

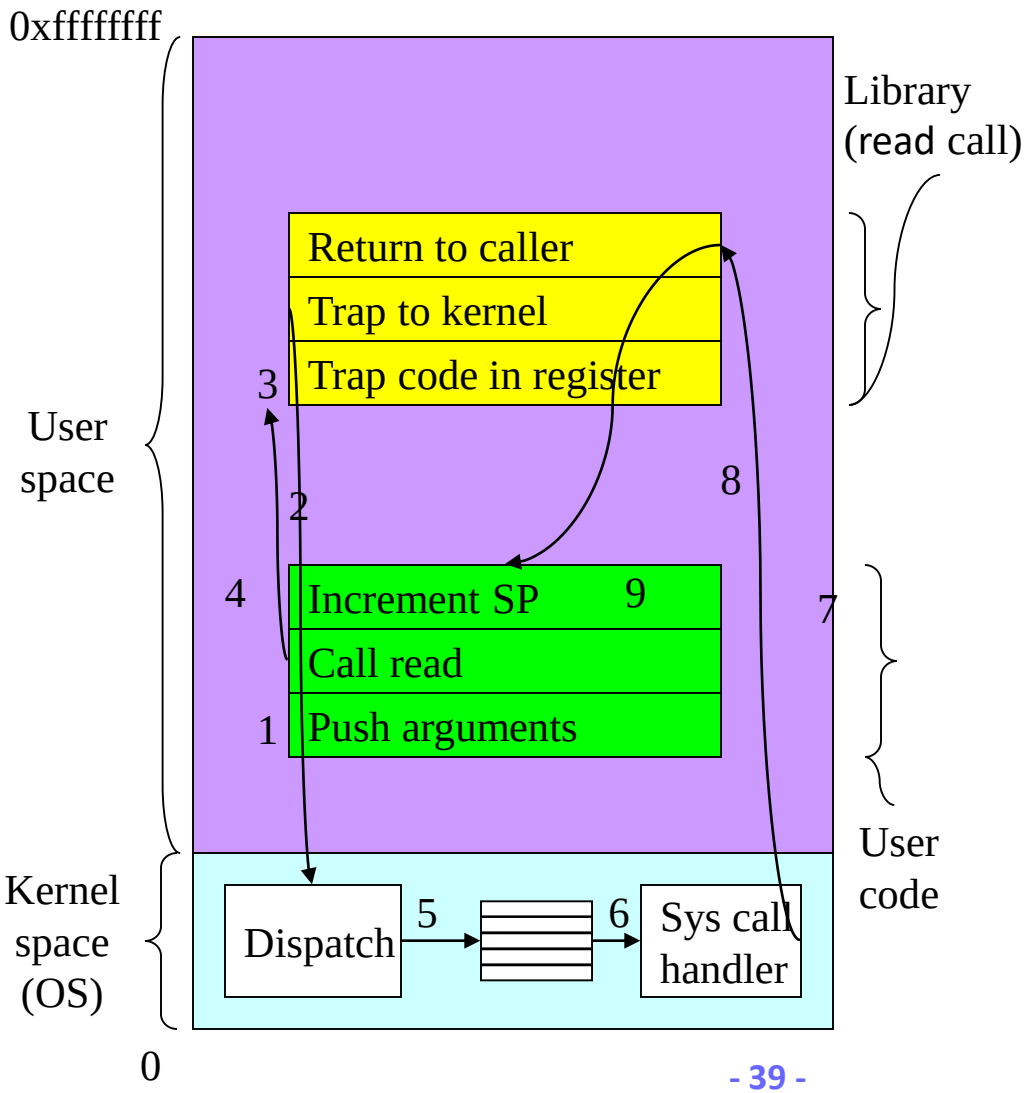
- Processes want to exchange information with each other
- Many ways to do this, including
  - Network
  - Pipe (special file): A writes into pipe, and B reads
  - Shared memory: everybody can read and write



System calls

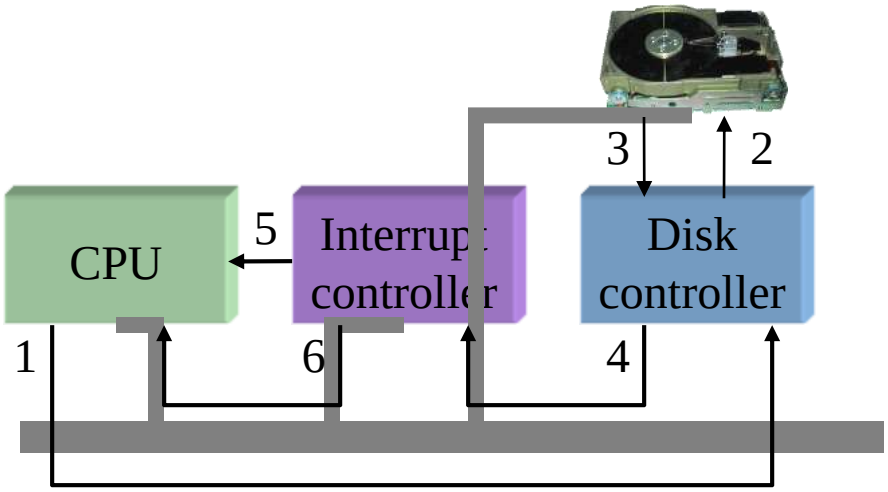
- **Programs want the OS to perform a service**
  - Access a file
  - Create a process
  - Read from network
  - …
- **Accomplished by system call**
  - Program passes relevant information to OS
  - OS performs the service if:
    - The OS is able to do so
    - The service is permitted for this program at this time
  - OS checks information passed to make sure it's OK
    - Don't want programs reading data into other programs' memory!

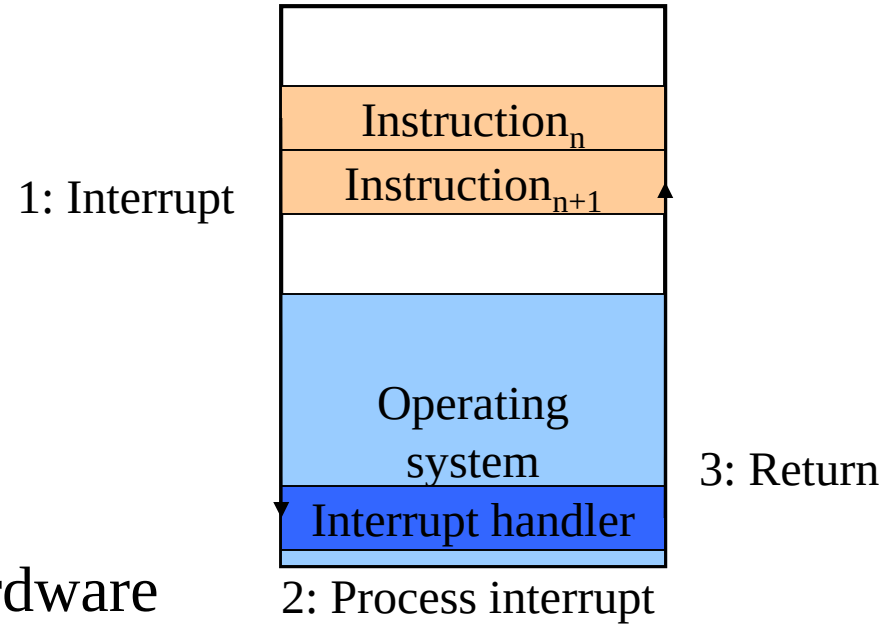
Making a system call



- System call:
  read(fd,buffer,length)
- Program pushes arguments, calls library
- Library sets up trap, calls OS
- OS handles system call
- Control returns to library
- Library returns to user program

Anatomy of a device request





- Left: sequence as seen by hardware
  - Request sent to controller, then to disk
  - Disk responds, signals disk controller which tells interrupt controller
  - Interrupt controller notifies CPU
- Right: interrupt handling (software point of view)

Interrupts

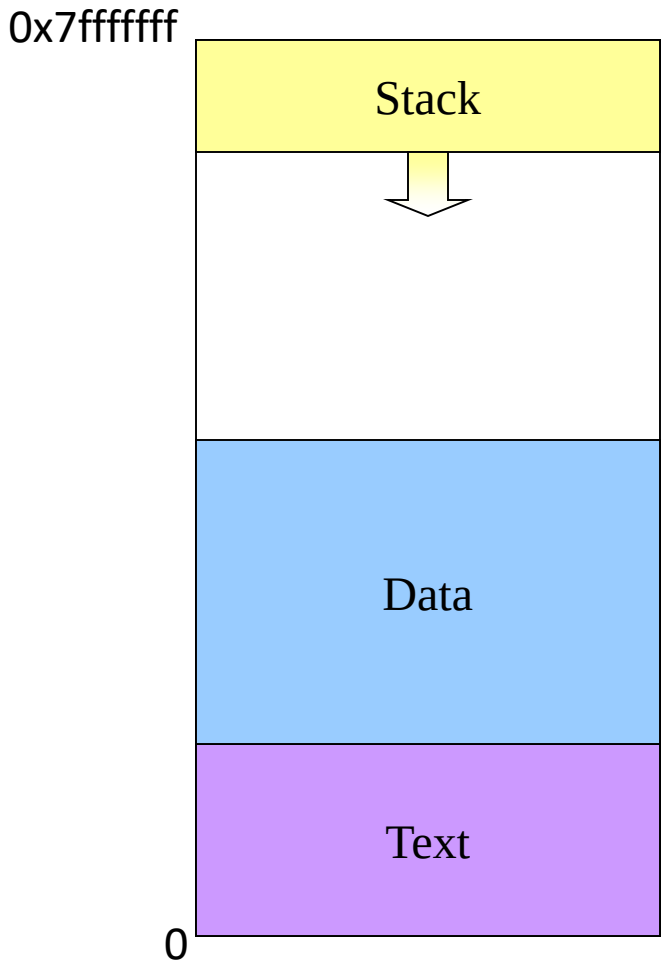
- The interrupt handler must save the machine state, do some processing, then call the process scheduler and dispatcher.

- When an interrupt occurs

  - the processor hardware makes a quick copy of the program counter and CPU registers
  - the hardware switches to kernel mode and jumps to the interrupt service routine
  - the ISR is usually very short. It may inform a device driver that it received the interrupt; it may just increment some clock counters.
  - next the ISR calls the scheduler, which decides which process to run
  - the scheduler calls the dispatcher, and new process (or maybe the same process) resumes where it left off

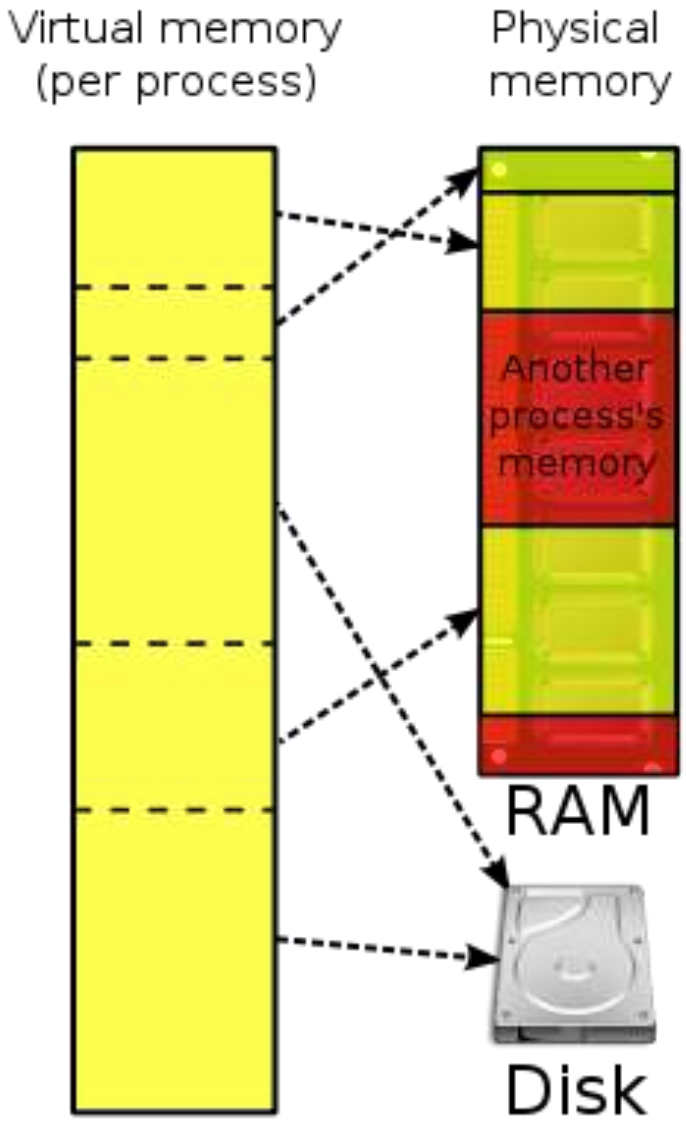
- An important goal of the OS is to hide interrupts from the user---and from user-level processes.

Inside a (Unix) process

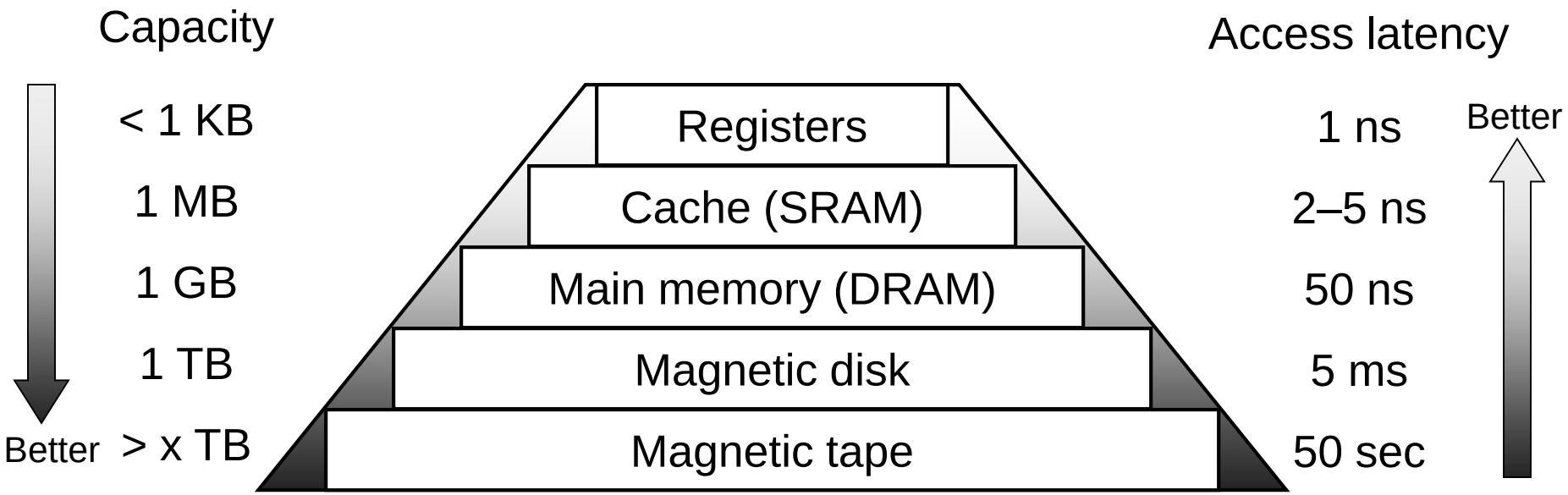


- Processes have three segments
  - Text: program code
  - Data: program data
    - Statically declared variables
    - Areas allocated by malloc() or new
  - Stack
    - Automatic variables
    - Procedure call information
- Address space growth
  - Text: doesn't grow
  - Data: grows "up"
  - Stack: grows "down"

Virtuaalmälu: OS mapib reaalse mälu „näilikuks“



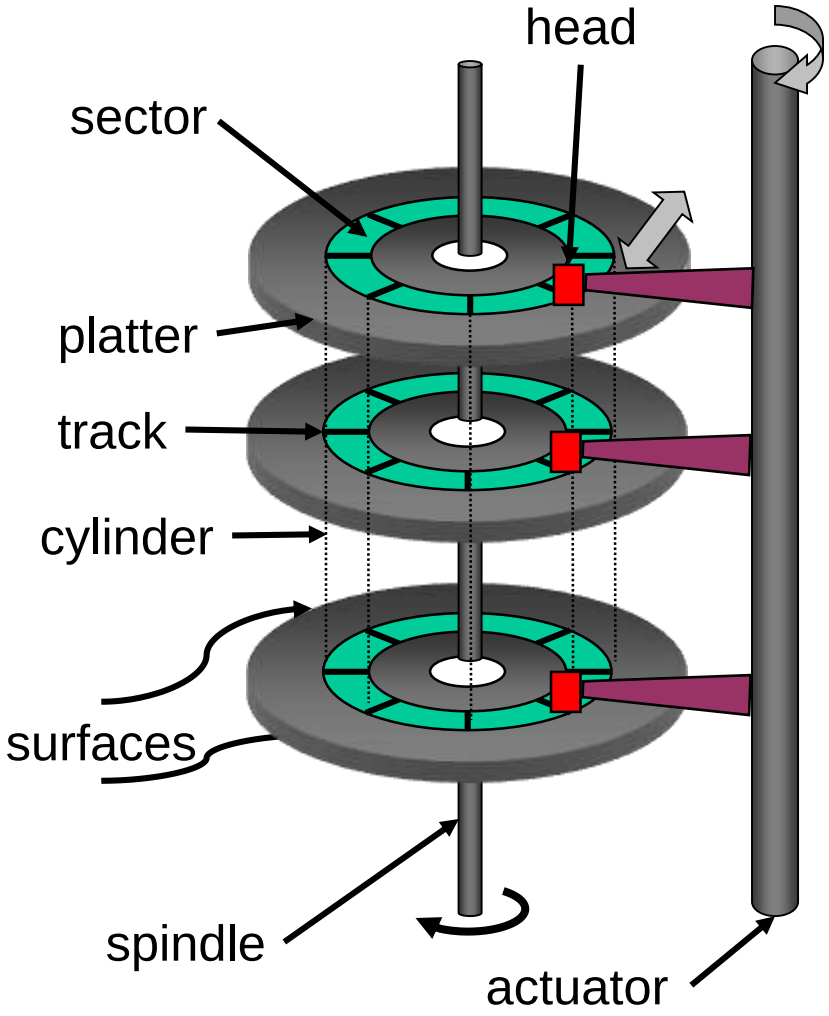
Storage pyramid



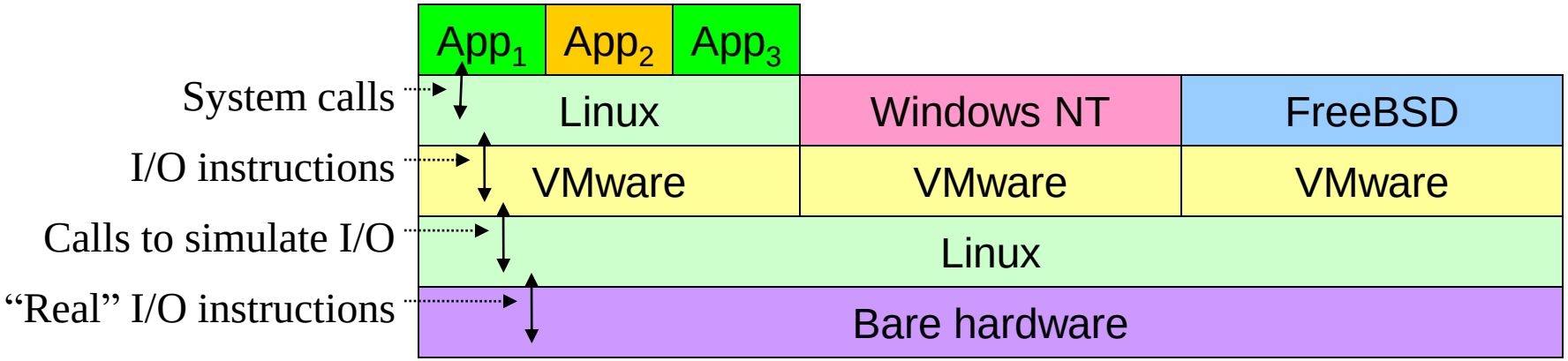
- Goal: really large memory with very low latency
  - Latencies are smaller at the top of the hierarchy
  - Capacities are larger at the bottom of the hierarchy
- Solution: move data between levels to create illusion of large memory with low latency

Disk drive structure

- Data stored on surfaces
  - Up to two surfaces per platter
  - One or more platters per disk
- Data in concentric tracks
  - Tracks broken into sectors
    - 256B-1KB per sector
  - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
  - Actuator moves heads
  - Heads move in unison

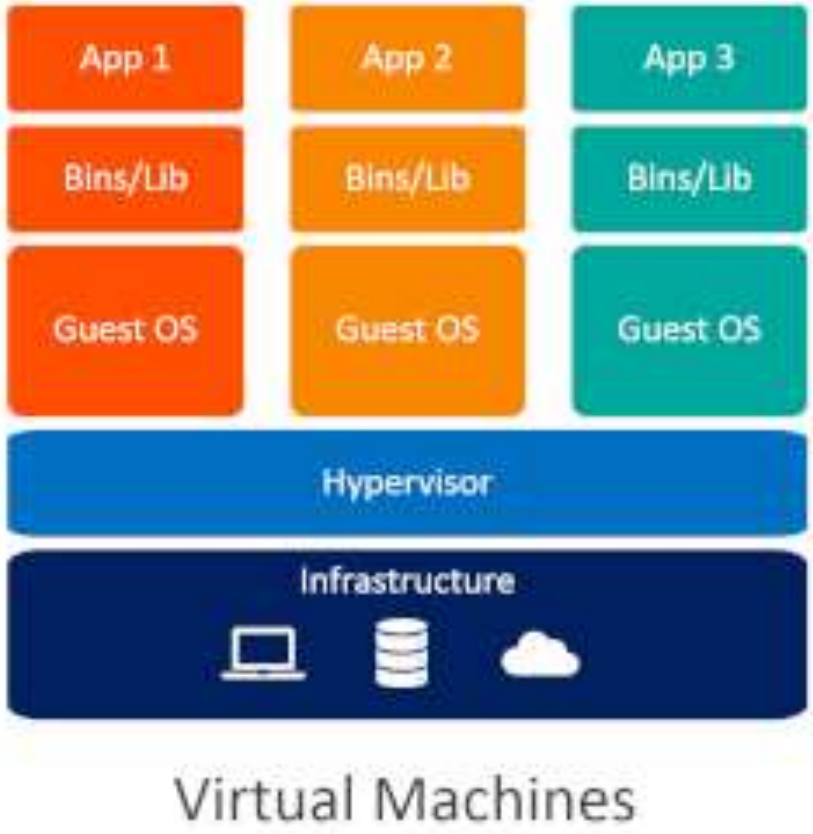


Virtual machines



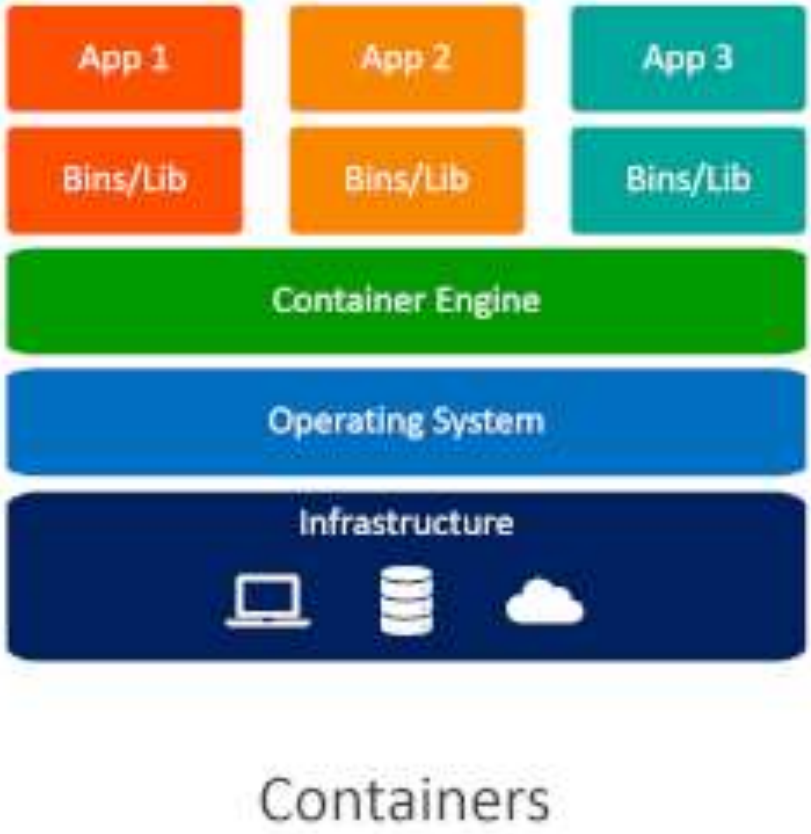
Virtuaalmasinad ja konteinerid

Vmware, VirtualBox, hyper-v



Virtual Machines

Docker



Containers